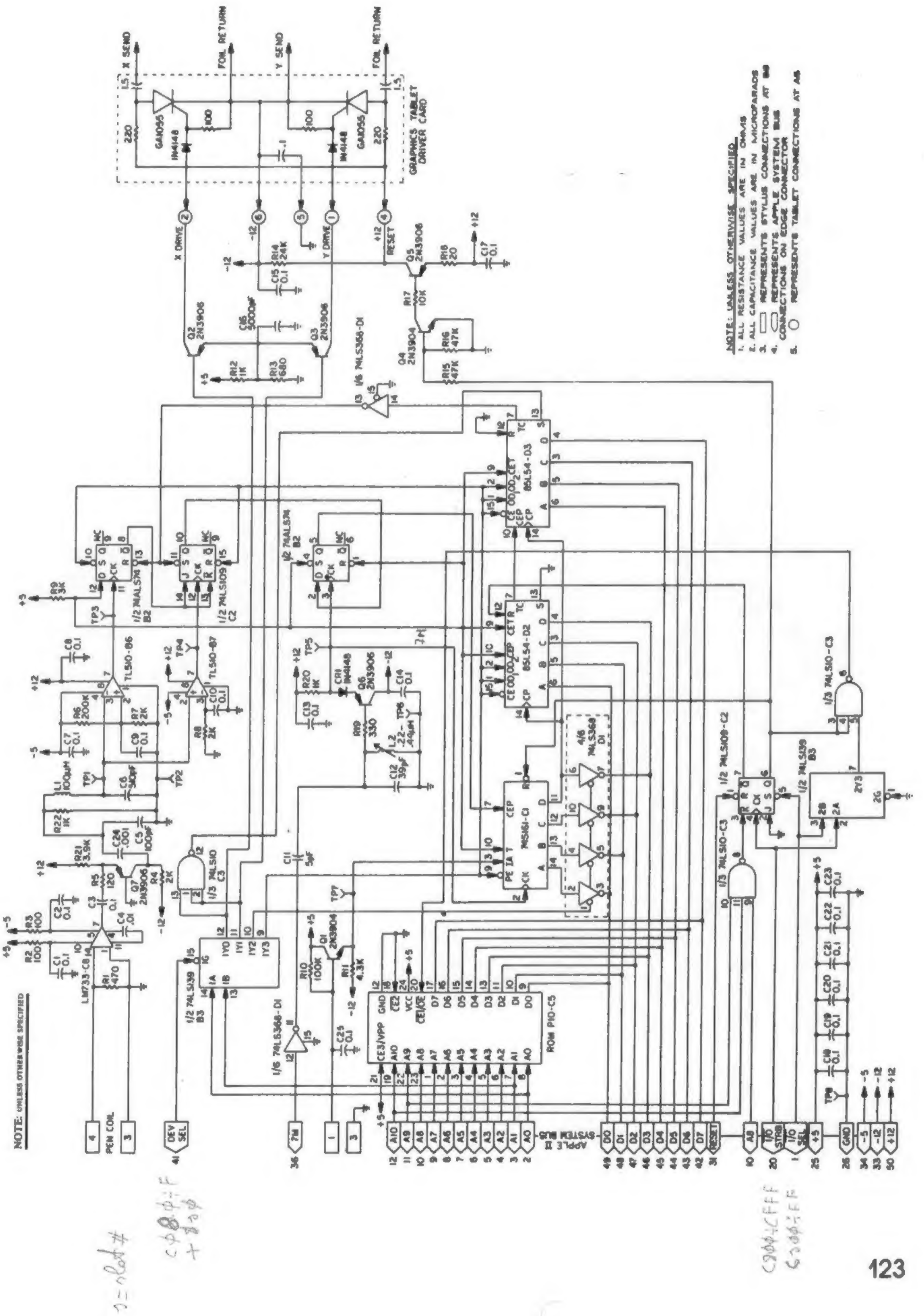
APPENDIX E
SCHEMATIC DIAGRAM



NOTE: UNLESS OTHERWISE SPECIFIED
1. ALL RESISTANCE VALUES ARE IN OHMS
2. ALL CAPACITANCE VALUES ARE IN MICROFARADS
3. ▭ REPRESENTS STYLUS CONNECTIONS AT [illegible]
4. ⬭ REPRESENTS APPLE SYSTEM BUS CONNECTIONS ON EDGE CONNECTOR
5. ○ REPRESENTS TABLET CONNECTIONS AT A5

n = slot #

C0n0-C0nF + $80

C800-CFFF
Cn00-CnFF

Chapter 8 Apple Desktop Bus

The Apple Desktop Bus (ADB) is a serial bus used for input devices such as keyboards, mouse devices, and graphics tablets. The ADB is Apple's standard interface for input devices; it is used on the Apple IIGS as well as the following Macintosh models:

- Macintosh SE
- Macintosh SE/30
- Macintosh II
- Macintosh IIx
- Macintosh IIcx

◆ *Note:* The Macintosh Plus and earlier Macintosh models do not include the Apple Desktop Bus. The interfaces to the mouse and the keyboard on those models are described in Chapter 7.

In addition to the ADB interface, this chapter describes the Apple Standard Mouse, the Apple Standard Keyboard, and the Apple Extended Keyboard. This chapter also describes the protocols and signals used on the ADB for communications between the computer and ADB devices, and the internal registers used by ADB devices.

△ **Important** Apple Computer, Inc. owns patents on the Apple Desktop Bus (ADB). If you wish to make a device that interfaces with the Apple Desktop Bus software, you must obtain a license and Device Handler ID from Apple Computer, Inc. Write to this address:
Apple Software Licensing
Apple Computer, Inc.
20525 Mariani Avenue, Mail Stop 28B
Cupertino, California 95014 △

	Mouse & keyboard
MacPlus	Macintosh
Mac SE	ADB
Mac SE/30	ADB
Mac II	ADB
Mac IIx	ADB
Mac IIcx	ADB

Overview

The Apple Desktop Bus is a single-master, multislave serial bus used to communicate with up to 16 low-speed input devices such as keyboards, mouse devices, and graphics tablets.

The Macintosh Apple Desktop Bus hardware consists of the following components:

- ADB transceiver IC, a 4-bit microcontroller that drives the bus and reads the status of the ADB devices
- VIA IC that provides the interface between the ADB transceiver IC and the computer's CPU (an MC68000, MC68020, or MC68030, depending on the model)
- two ADB 4-pin connectors, hooked up in parallel. The ADB connectors are located on the rear panel of the computer and are used for attaching ADB input devices

The ADB hardware provided by peripheral devices on the ADB network consists of the following components:

- ADB transceiver IC in each ADB peripheral device
- Four-conductor shielded cable used to connect the ADB device to the ADB

△ **Important** Although the ADB is capable of addressing up to 16 different peripheral devices, daisy-chaining more than 3 devices is not recommended because of connector resistance and signal degradation. △

The ADB transceiver IC in the computer converts the computer's TTL signals to the variable-pulse-width open-collector signal used on the bus. When the bus is idle, the transceiver performs automatic polling of the ADB device that last sent data.

The computer transfers data to and from the ADB transceiver IC by using the Shift register in the VIA to send and receive the TTL signals. The Shift register contains the 8 data bits that have been shifted in from the ADB transceiver, or the 8 bits that are to be shifted out. In addition to the ADB data line, there are several control lines between the VIA and the ADB transceiver. These control lines are used to time and sequence transactions between the VIA and the ADB transceiver.

◆ *Note:* On Macintosh models that have two VIA ICs, VIA1 is the interface to the ADB.

When the computer is started up or reset, each ADB peripheral device defaults to an ADBdevice address from $00 through $0F. Certain default addresses are used by specific device types; for example, relative-position devices (such as a mouse device) always have a default ADB address of $03. In the case of two or more devices with the same default address, the Start Manager assigns a new (unique) address to one device at a time until all conflicts are eliminated.

▲ **Warning** Do not unplug and reattach an ADB device while the computer is running—if you do, the device reverts to its default address while the computer continues trying to reach it at the address assigned at startup time. ▲

Each ADB device has a default identification code known as a *Device Handler ID*, stored in an internal register. The ADB Manager keeps track of the Device Handler ID and the default address of each device on the bus, and calls the appropriate device driver when that device has data to send to the computer.

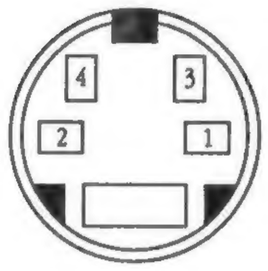
An ADB device cannot initiate a data transaction. Instead, each device asserts a Service Request signal when it has data to send to the computer. The ADB Manager then attempts to read each device on the bus until one responds by sending data. When a device responds, the ADB Manager passes control to the appropriate device driver, which communicates with the device through the VIA and the ADB transceiver.

The ADB communication protocol and the functions of an ADB device's internal registers are described in detail in the sections "ADB Communications" and "ADB Peripheral Devices," later in this chapter. For more information on the VIA, see Chapter 4.

ADB interface

The ADB connector pinouts are shown in Figure 8-1. The ADB connector signal assignments are listed in Table 8-1. The electrical characteristics of the ADB data line and of ADB devices are listed in Table 8-2.

■ **Figure 8-1** Pinouts for the ADB connector



■ **Table 8-1** ADB connector signal assignments

Pin Number	Signal name	Signal description
1	ADB	Bidirectional data bus used for input and output. It is an open-collector signal pulled up to +5 V through a 470 Ω resistor on the computer's main logic board.
2	Power On	On the Macintosh II family, a key on the ADB keyboard momentarily grounds this pin to pin 4 to switch on the power supply. On other models, this pin is not connected.
3	+5V	+5 volts
4	GND	Ground

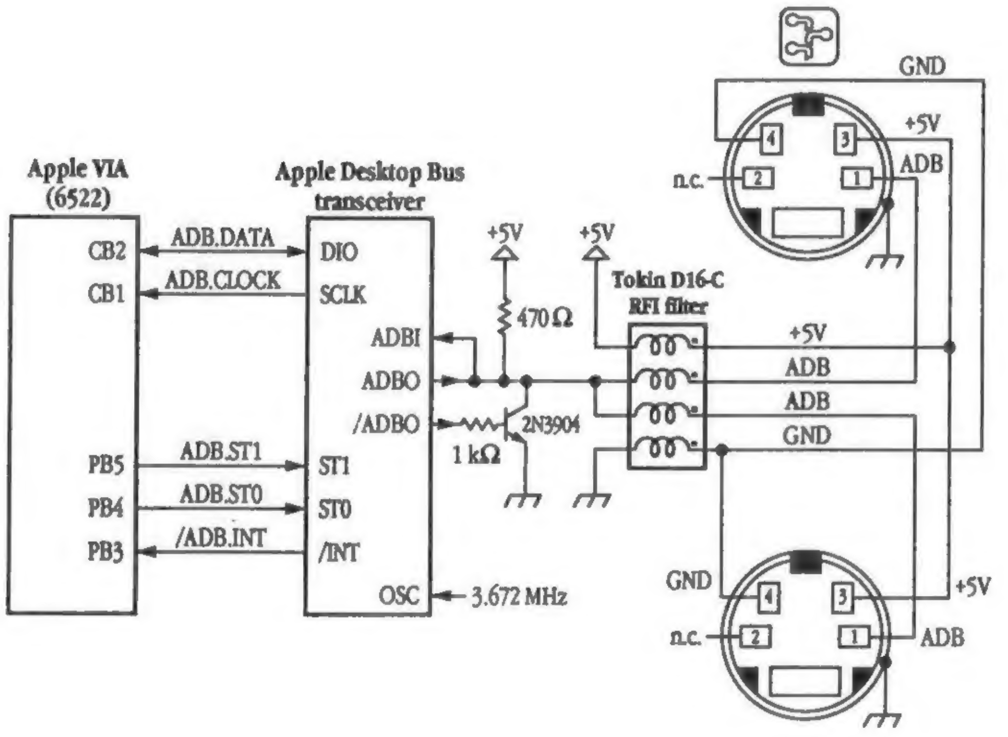
■ **Table 8-2** Electrical characteristics of the ADB

Parameter	Minimum	Maximum
Low input signal voltage	–0.2 V	0.8 V
High input signal voltage	2.4V	5.0V
Low output signal voltage	—	0.45 V (at 12 mA)
High output signal voltage	2.4 V	—
Device output current when off	—	–20 μA (at 0.4 V)
Device input capacitance	—	150 pF

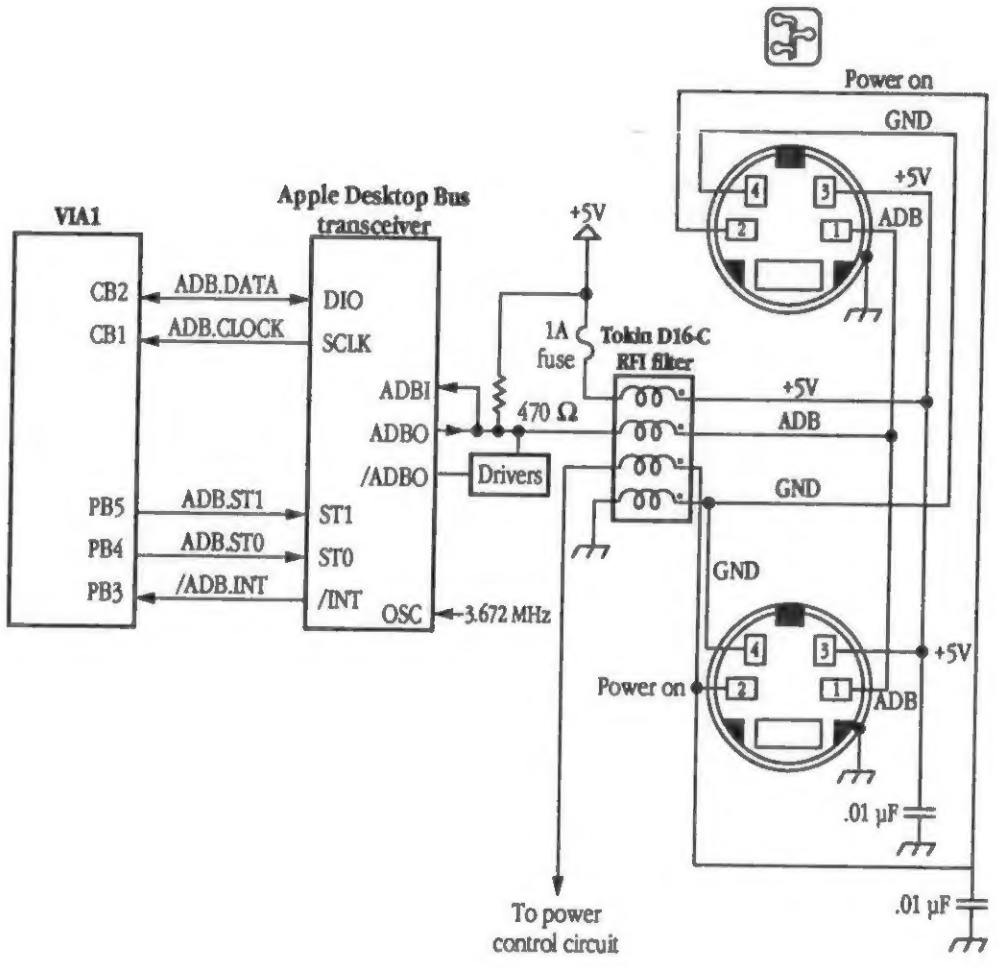
ADB interface circuits

The ADB interface is functionally the same on all Macintosh models except the Macintosh Plus and earlier models, which don't use it. Figure 8-2 shows a circuit diagram for the ADB interface on the Macintosh models that do not have the keyboard power-on feature (Macintosh SE and Macintosh SE/30). Figure 8-3 shows a circuit diagram for the ADB interface on Macintosh models that do have the keyboard power-on feature (the Macintosh II, Macintosh IIx, and Macintosh IIcx).

■ **Figure 8-2** Circuit diagram for the ADB interface on Macintosh SE and Macintosh SE/30



■ **Figure 8-3** Circuit diagram for the ADB interface on Macintosh II, Macintosh IIx, and Macintosh IIcx computers



△ **Important** ADB devices may use the +5 volts supplied by the bus, but all the ADB devices combined must not draw more than a total of 500 mA. △

Cables should be no longer than 5 meters, and cable capacitance should not exceed 100 picofarads per meter. Although the ADB is capable of addressing up to 16 different peripheral devices, daisy-chaining more than 3 devices on one ADB port is not recommended because of connector resistance and signal degradation.

△ **Tip** The ADB device protocol and the device control commands are the same on all ADB-equipped Macintosh computers. However, different models may use different hardware to implement the ADB. To ensure compatibility with all Macintosh computers, it is important that you use the routines and tools provided at the system level for controlling the ADB, as discussed in *Inside Macintosh.* △

ADB keyboards and mouse

There are two ADB keyboard options available from Apple Computer, Inc.: the Apple Standard Keyboard and the Apple Extended Keyboard. Apple also provides an Apple Standard Mouse device with each ADB-equipped Macintosh computer. The mouse and keyboards communicate with the main logic board by way of the ADB interface.

△ **Tip** Your applications should read all keyboard and mouse data by using the Event Manager and Window Manager so that your program will be compatible with earlier Macintosh computers, which do not use the ADB, and with future models, which may use different ADB hardware. The Event Manager and the Window Manager are described in *Inside Macintosh.* △

Apple Standard Mouse

The Apple Standard Mouse is used by all Apple computers that have the Apple Desktop Bus.

A microprocessor in the Apple Standard Mouse provides position and status information to the computer. The mouse mechanism reports relative X (left and right) and Y (up and down) motion and indicates when the mouse button is pressed. This microprocessor also functions as an ADB transceiver chip, which communicates with the ADB transceiver chip in the computer. The computer's ADB transceiver chip receives the mouse information and sends it to the VIA. Routines in the ROM convert this information into the corresponding motion of the pointer on the screen.

The mouse operates by sending square-wave signals to the mouse microprocessor as the mouse moves. A rubber-coated steel ball in the mouse turns as the mouse is moved and contacts two capstans, each connected to a slotted wheel called an *interrupter wheel.* Motion in the mouse's X-axis direction rotates one of the wheels and motion in the Y-axis direction rotates the other wheel.

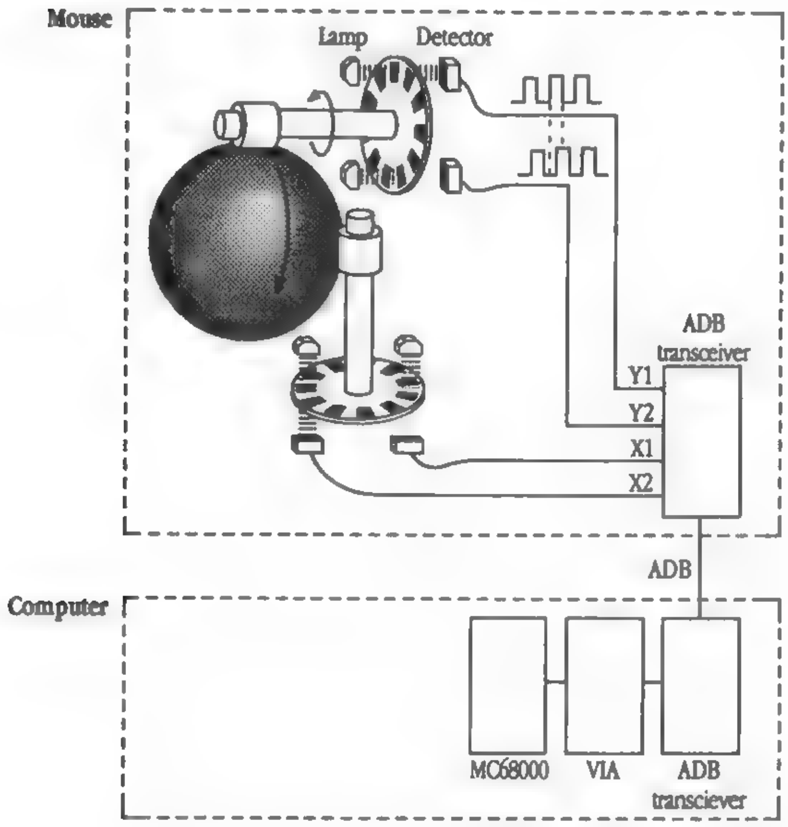
Beams of light are alternately transmitted and blocked by the slots in the interrupter wheel, and photo detectors pass the resulting square-wave signal to the mouse microprocessor. The microprocessor interprets these signals to determine the distance travelled in the X and Y directions and passes the information to the Mouse Driver software by way of the ADB transceiver and the VIA.

The mouse mechanism uses a scheme known as *quadrature* to determine the direction the mouse is moving along each axis. Two beams of infrared light shine through the slots in the interrupter wheel and strike one of two light detectors. The detectors are offset just enough so that, as the wheel turns, they produce two square-wave signals that are 90° out of phase. These signals are called the *interrupt signal* and the *quadrature signal.* The quadrature signal precedes the interrupt signal by 90° when the wheel turns one way, and trails it when the wheel turns the other way.

◆ *Note:* The mouse used in the Macintosh Plus and earlier Macintosh computers sends the square-wave signals generated by the interrupter wheels directly to the computer, where they are interpreted by the Mouse Driver software. The Macintosh Plus mouse is described in Chapter 7, "Macintosh Plus Mouse and Keyboard."

Figure 8-4 shows the Apple Standard Mouse mechanism. The square-wave signals in this figure are those generated when the mouse is moved downward.

■ **Figure 8-4** Apple Standard Mouse mechanism



The mouse motion information is stored in register 0 in the ADB transceiver IC in the mouse. (The registers in peripheral devices are described in the section "ADB Device Registers," later in this chapter.) The contents of ADB register 0 in the Apple Standard Mouse are shown in Table 8-3.

■ **Table 8-3** Apple Standard Mouse ADB transceiver register 0

Bit	Meaning
15	Button status; 0 = down
14–8	Y move counts*
7	Not used (always 1)
6–0	X move counts**

*Two's-complement form. Negtive = up, positive = down.

**Two's-complement form. Negtive = left, positive = right.

◆ *Note:* For a Device Handler ID of $0001, the mouse accumulates 100±10 counts on each axis for each inch of travel in that axis. For a Device Handler ID of $0002, the mouse accumulates 200±10 counts on each axis for each inch of travel in that axis. On startup or reset, the mouse has a Device Handler ID of $0001.

△ **Tip** The transfer of mouse data is handled automatically by the ADB Manager and the Mouse Driver. To ensure compatibility with all Macintosh computers, your software should always use Event Manager and Window Manager calls to determine mouse motion and the state of the mouse button. △

The ADB transceiver in the mouse transmits the data to the ADB transceiver in the computer, which transmits it to the Shift register in the VIA. The Mouse Driver reads the information in the VIA's Shift register, interprets the data, and translates it into cursor motion on the screen as appropriate. Using the mouse-tracking option on the Control Panel desk accessory, the user can change the amount of screen-pointer motion that corresponds to a given mouse motion. For more information about the rate of mouse motion, see the discussion of parameter RAM in the chapter on Operating System Utilities in *Inside Macintosh.*

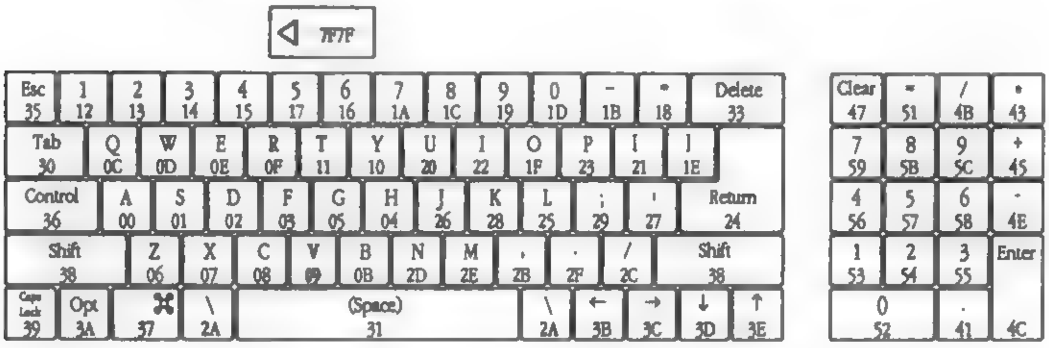
Apple Standard Keyboard

The switches in the keyboard are wired in a two-dimensional array called a *keyswitch matrix.* Like the keyboard on the Macintosh Plus, the Apple Standard Keyboard contains a microprocessor that scans the matrix to detect switch transitions as keys are pressed and released. The keyboard's microprocessor transmits the corresponding key-down and key-up events to the CPU by way of the Apple Desktop Bus (ADB), using the bus protocols described later in this chaper.

In addition to all the keys found on the Macintosh Plus keyboard (or their functional equivalents), the Apple Standard Keyboard has two additional keys: the Escape key and the Control key.

The keyboard layout and the key-down transition codes are shown in Figure 8-5. The key-up transition codes are the same as the key-down transition codes, except that bit 7 is set to 1.

■ **Figure 8-5** Key-down transition codes generated by the Apple Standard Keyboard



Notice that these response codes are not all the same as the key codes returned by the Keyboard Driver software. The standard 'KMAP' resource supplied in the System Folder reassigns several key codes, as shown in Table 8-4.

■ **Table 8-4** Apple Standard Keyboard key code reassignments

Key	Transition code	Keyboard Driver code
Control	$36	$3B
Left Arrow	$3B	$7B
Right Arrow	$3C	$7C
Down Arrow	$3D	$7D
Up Arrow	$3E	$7E

The keyboard information is stored in register 0 in the ADB transceiver in the Apple Standard Keyboard. The statuses of two keys can be stored at the same time in Keyboard register 0. In addition, the statuses of the modifier keys are stored in register 2 in the keyboard. The registers in peripheral devices are described in the section "ADB Device Registers," later in this chapter. The contents of Keyboard register 0 are shown in Table 8-5 and the contents of Keyboard register 2 are shown in Table 8-6.

■ **Table 8-5** Apple Standard Keyboard register 0

Bit	Meaning
15	Key status for first key; 0 = down
14–8	Key code for first key; a 7-bit ASCII value
7	Key status for second key; 0 = down
6–0	Key code for second key; a 7-bit ASCII value

■ **Table 8-6** Apple Standard Keyboard register 2

Bit	Key
15	None (reserved)
14	Delete
13	Caps Lock
12	Reset
11	Control
10	Shift
9	Option
8	Command
7–0	None (reserved)

Note: A 0 indicates that a key is down.

Upon a command from the ADB Manager, the ADB transceiver in the Apple Standard Keyboard transmits the data in register 0 to the ADB transceiver in the computer, which transmits it to the VIA's Shift register. The ADB Manager then passes control to the Keyboard Driver, which reads all the keyboard information, interprets it, and makes it available to applications.

The Apple Extended Keyboard

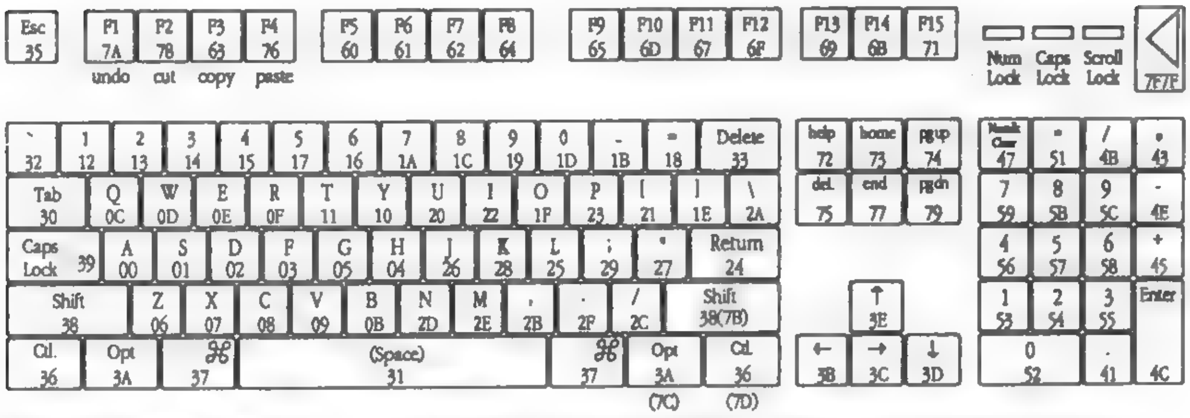
The Apple Extended Keyboard is designed primarily for users who wish to run operating systems other than the Macintosh system. This keyboard is also useful with applications programs and data communications packages ported from other computers that use function keys.

The switches in the keyboard are wired in a two-dimensional array called a *keyswitch matrix*. Like the Apple Standard Keyboard, the Apple Extended Keyboard contains a microprocessor that scans the matrix to detect switch transitions as keys are pressed and released and transmits the corresponding key-down and key-up events to the CPU by way of the Apple Desktop Bus (ADB), using the bus protocols described later in this chaper.

Like the Apple Standard Keyboard, the Apple Extended Keyboard has all the keys (or their functional equivalents) found on earlier Macintosh keyboards, plus the modifier keys Escape (Esc) and Control. In addition, the Apple Extended Keyboard has function keys: twelve of the function keys (F1–F12) have no default definition; nine other function keys do have default definitions.

The Apple Extended Keyboard layout and key-down transition codes are shown in Figure 8-6. The key-up transition codes are the same as the key-down transition codes except that bit 7 is set to 1.

■ **Figure 8-6** Key-down transition codes generated by the Apple Extended Keyboard



If your application needs to be able to distinguish the right-hand Option and Control keys from the corresponding keys on the left side of the keyboard, you can cause the keyboard to generate different transition codes for those keys (shown in parentheses in Figure 8-6) by changing the value of the Device Handler ID in Keyboard register 3 from $0002 to $0003. You can change this ID by sending a Listen Register 3 command to the keyboard. For more information, see the section "ADB Device Handler ID," later in this chapter, and the description of the Device Handler ID command in the chapter on the Apple Desktop Bus in *Inside Macintosh*.

Note that the transition codes shown in Figure 8-6 are not all the same as the key codes returned by the Keyboard Driver software. The standard 'KMAP' resource supplied in the System Folder reassigns several key codes, as shown in Table 8-7.

■ **Table 8-7** Apple Extended Keyboard key code reassignments

Key	Transition code	Keyboard Driver code
Control	$36	$3B
Left Arrow	$3B	$7B
Right Arrow	$3C	$7C
Down Arrow	$3D	$7D
Up Arrow	$3E	$7E
Right Shift	$7B*	$3C*
Right Option	$7C*	$3D*
Right Control	$7D*	$3E*

*These key codes are in effect only when the Device Handler ID in register 3 is set to $0003.

The keyboard information is stored in register 0 in the ADB transceiver in the Apple Extended Keyboard. The statuses of two keys can be stored at one time in keyboard register 0. In addition, the statuses of the modifier keys and LEDs are stored in register 2 in the keyboard. The registers in peripheral devices are described in the section "ADB Device Registers," later in this chapter. The contents of Keyboard register 0 are shown in Table 8-8 and the contents of Keyboard register 2 are shown in Table 8-9.

■ **Table 8-8** Apple Extended Keyboard register 0

Bit	Meaning
15	Key status for first key; 0 = down
14–8	Key code for first key; a 7-bit ASCII value
7	Key status for second key; 0 = down
6–0	Key code for second key; a 7-bit ASCII value

■ **Table 8-9** Apple Extended Keyboard register 2

Bit	Meaning
15	None (reserved)
14	Delete
13	Caps Lock
12	Reset
11	Control
10	Shift
9	Option
8	Command
7	Num Lock/Clear
6	Scroll Lock
5–3	None (reserved)
2	LED 3 (Scroll Lock)*
1	LED 2 (Caps Lock)*
0	LED 1 (Num Lock)*

*You can change the value of this bit with a Listen Register 2 command.

Note: A zero indicates that a key is down or that an LED is on.

Upon a command from the ADB Manager, the ADB transceiver in the Apple Extended Keyboard transmits the data in register 0 to the ADB transceiver in the computer, which transmits it to the VIA's

Shift register. The ADB Manager then passes control to the Keyboard Driver, which reads all the keyboard information, interprets it, and makes it available to applications.

ADB communications

Each ADB input device on the network may have up to four internal registers, which are referred to as *ADB device registers*. ADB devices are accessed over the network by reading from or writing to the appropriate ADB device register in that device. The registers of an ADB device may range in size from 2 bytes to 8 bytes.

Although each ADB device contains a microprocessor and can generate device identification and status information, an ADB device cannot initiate a command; nor can it interrupt the processor directly. The ADB transceiver in the computer automatically polls the last device to have sent data (known as the *active device*) for new data. If the active device has data to send, it shifts it out to the ADB transceiver in the computer, which sends it to the VIA. When the VIA receives ADB data, a bit in the VIA's Interrupt register is set. The ADB Manager can thus determine that the active device has sent data. The ADB Manager then passes control to the device driver for the active device, which communicates with the device through the VIA and the ADB transceiver.

◆ *Note:* On Macintosh models that have two VIA ICs, VIA1 is the interface to the ADB.

If an ADB device that is not the active device has data to send, the device asserts a Service Request signal to the computer. The ADB transceiver in the computer asserts its interrupt request signal to the VIA, which sets bit 3 in VIA's Data register B to 0. The Macintosh Operating System polls the VIA regularly; when it finds a 0 for this bit, the operating system passes control to the ADB Manager, which polls each device until it finds the one requesting service. The ADB Manager then passes control to the appropriate device driver, which communicates with the device through the VIA and the ADB transceiver.

Communications on the ADB are of two types: transactions and signals. Transactions consist of a command from the computer to the ADB device, followed by data sent by either the computer or the device. Each command is addressed to a specific device. Signals indicate the status of a device or cause all devices to perform a certain function. Signals are general; they are not addressed to a specific device.

The next sections define ADB transactions, describe each of the commands and signals, and list the timing specifications for ADB commands and signals.

ADB transactions

An ADB transaction is a bus communication between the computer and a device. A transaction consists of a command sent by the computer, followed by a data packet of several bytes sent by either the computer or a device. .An ADB command consists of

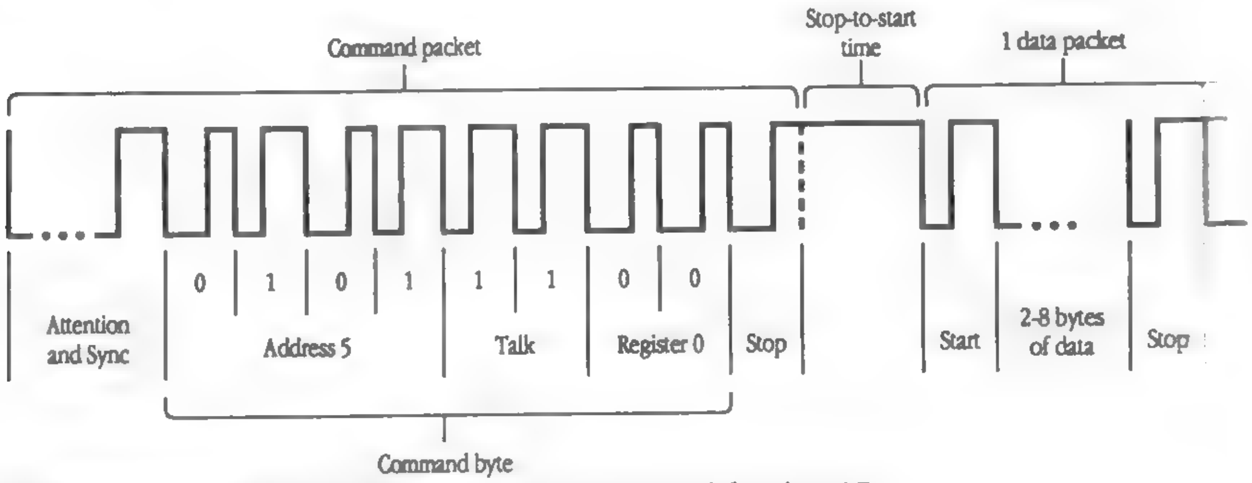
- an Attention signal
- a Sync signal
- one command byte
- one stop bit

A data packet consists of

- a start bit
- two to eight (8-bit) data bytes
- one stop bit

Figure 8-7 shows a typical transaction on the Apple Desktop Bus, consisting of a command followed by a data packet.

- **Figure 8-7** A typical ADB transaction



Two signal lines from the VIA to the transceiver—ST0 and ST1—define the ADB transaction state, which controls the sequence of operations performed by the ADB transceiver chip. After each part of an ADB command transaction is complete, the computer uses lines ST0 and ST1 to advance the ADB transceiver to the next transaction state, ready for the next part of the communication. There are four possible ADB transaction states, as shown in Table 8-10.

■ **Table 8-10** ADB transaction states

Signals		
ST1	**ST0**	**Transaction state**
0	0	0: Start a new command
0	1	1: Transfer data byte (even)
1	0	2: Transfer data byte (odd)
1	1	3: Idle

To execute an ADB command, the ADB Manager sends the command byte to the VIA's Shift register, then sets the ADB transceiver to state 0. The transceiver shifts the command in from the VIA, converts it to ADB protocol, and puts it on the ADB. The VIA generates an interrupt when the entire byte has been transferred. When the processor receives the interrupt, the ADB Manager alternates the transaction state between 1 and 2 to transfer the data bytes from the VIA to the ADB transceiver and then over the ADB (for a Listen command), or from the device to the transceiver and then into the VIA (for a Talk command). (The Talk and Listen commands are described in the next section, "ADB Commands.") The VIA generates an interrupt after each byte is transferred.

After the last byte of data has been transferred, the ADB Manager sets the transaction state to 0 if the ADB Manager wants to send another command, or to 3, in which state the ADB transceiver automatically repeats the last Talk command every 11 ms. To abort a partially executed command, the ADB Manager sets the ADB transaction state to 0.

The default transaction state on startup or reset is 3.

ADB commands

An ADB command is sent by the computer to one specific device address. There are four commands:

- Talk
- Listen
- SendReset
- Flush

A command is an 8-bit word that has a specific syntax as shown in Table 8-11. Every command consists of

- a 4-bit field that specifies the address of the desired device
- a 2-bit command code

- a 2-bit register code

◆ *Note:* Although only the computer can initiate a command, the ADB transceiver can reissue a Talk command when the device addressed does not respond and there is no service request pending from another device. Therefore, the ADB Manager does not have to use processor time to monitor the bus continually.

■ **Table 8-11** Command byte syntax

	Device address				Command code		Register code		Command
Bit	7	6	5	4	3	2	1	0	
	x	x	x	x	0	0	0	0	SendReset
	A3	A2	A1	A0	0	0	0	1	Flush
	x	x	x	x	0	0	1	0	Reserved
	x	x	x	x	0	0	1	1	Reserved
	x	x	x	x	0	1	x	x	Reserved
	A3	A2	A1	A0	1	0	r1	r0	Listen
	A3	A2	A1	A0	1	1	r1	r0	Talk

Note: x = ignored; r = register number; A3 through A0 = bits 11 through 8 of register 3.

△ **Tip** To allow for future expansion of the command structure, Apple has reserved several commands. Applications that use commands listed as reserved in Table 8-11 may have compatibility problems on future Apple products. △

Talk command

The Talk command initiates a data transfer to the computer from a specific register (0 through 3) of an ADB input device. When the computer sends a Talk command to a device, the device must respond with data within 260 µs. The selected device performs its data transaction and releases the bus, leaving it high. If the device does not respond in time (the device times out), the computer takes control of the bus again to issue its next command. A device will time out if it has no data to send; however, a device must respond to a Talk Register 3 command.

Listen command

The Listen command is a request for the device to receive data transmitted from the computer and store it in a specific internal register (0 through 3). When the computer sends a Listen command to a device, the device receives the next data packet from the computer and places it in the appropriate register. After the stop bit following the data is received, the transaction is complete and the computer releases the bus. If the addressed device detects another command on the bus (that is, receives an Attention signal and a Sync signal) before it receives any data, the original transaction is immediately considered complete.

SendReset command

The SendReset command causes all devices on the network to reset to their power-on states. Each device clears all pending service requests and returns to a state in which it can accept commands and assert Service Request signals.

Flush command

The action of the Flush command is defined for each device. Normally, it is used to clear any internal registers in the device. Any user input data being stored by the device—such as characters in a keyboard type-ahead buffer—are lost.

ADB Signals

There are four global signals used in ADB communications:

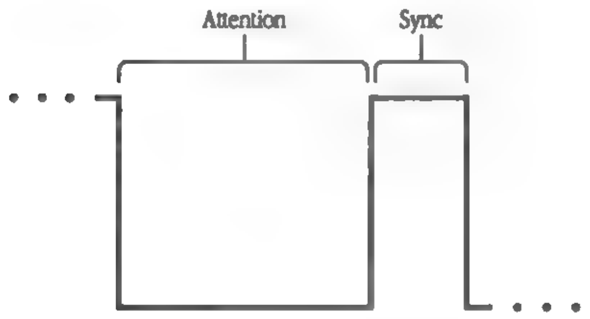
- Attention
- Sync
- Global Reset
- Service Request

Signals are transmitted on the bus but do not address any specific device. The first three of these signals are always generated by the computer. The Service Request signal is always generated by a device.

Attention and Sync signals

The start of every command is indicated by a long low Attention signal that the computer sends on the bus. This is followed by a short high Sync pulse that establishes the timing of the data bits that follow. The first command bit follows immediately after the falling edge of the Sync signal. Figure 8-8 shows the format of the Attention and Sync signals.

■ **Figure 8-8** Attention and Sync signals



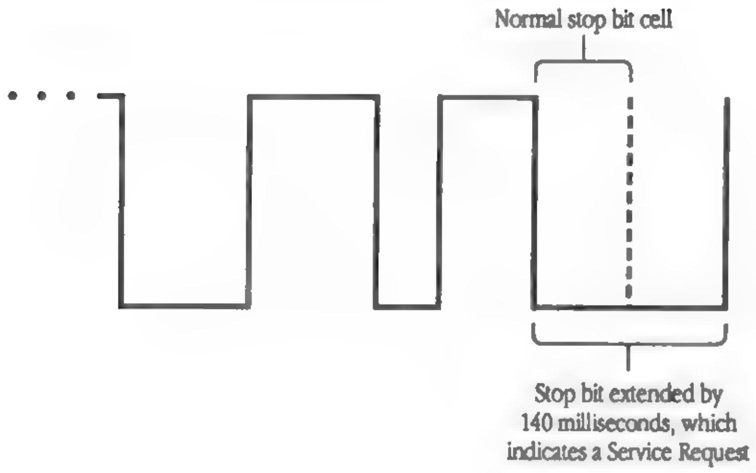
Global Reset signal

The computer initiates a reset of all devices on the ADB bus by holding the bus low for a minimum of 3.0 ms. Each device clears all pending service requests and returns to a state in which it can accept commands and assert Service Request signals.

Service Request signal

A Service Request signal is used by a device to inform the computer that the device has data to send. A device sends a Service Request signal by holding the bus low during the low portion of the stop bit of any command or data transaction. The device must lengthen the stop by a minimum of 140 µs beyond its normal duration, as shown in Figure 8-9.

■ **Figure 8-9** ADB Service Request signal


Normal stop bit cell
Stop bit extended by 140 milliseconds, which indicates a Service Request

A device sends a Service Request signal repeatedly until it receives a command from the computer. When a device requests service, the computer does not know which device sent the request. Therefore, the computer polls each of the devices by sending a Talk Register 0 command, beginning with the last active device. Only a device that has data to send responds to the Talk command.

When the computer sends a Talk command to the requesting device, the device is considered served and does not send a Service Request signal again until it needs to be served again. The computer can set a bit in register 3 to enable or disable the ability of a device to send a Service Request signal . (See the section "Register 3," later in this chapter.)

ADB timing

Each command or data bit is encoded on the ADB as a short pulse known as a *bit cell.* Each bit cell consists of a low voltage on the ADB, a rising edge, a high voltage on the ADB, and a final falling edge. A 0 is distinguished from a 1 by the relative length of the low time in the bit cell, as illustrated in Figure 8-7.

Every command and data packet ends with a stop bit. A stop bit is a 0 bit that has a low time as long as any other 0 bit but that does not necessarily have a second falling edge to define the end of the bit cell. Note that the time from the stop bit to the next start bit is critical; the computer requires this minimum turnaround delay to allow for internal overhead.

ADB signals are distinguished from command and data bits by having low and high times different from those used for bit cells. Table 8-12 lists the timing parameters for ADB bit cells and signals.

■ **Table 8-12** ADB timing specifications

Parameter	Nominal	Host	Device
Bit-cell time	100 µs	±3%	±30%
"0" low time	65 µs	65% of bit-cell time ±5%	65% of bit-cell time ±5%
"1" low time	35 µs	35% of bit-cell time ±5%	35% of bit-cell time ±5%
Attention low time	800 µs	±3%	—
Sync high time	65 µs	±3%	—
Stop bit low time	70 µs	±3%	±30%
Global Reset low time	3 ms	3 ms minimum	3 ms minimum
Service Request low time	300 µs	—	±30%
Stop bit to start bit time	200 µs	140 µs minimum 260 µs maximum	140 µs minimum 260 µs maximum

ADB error conditions

Two error conditions are defined for the ADB: when the bus remains low for an excessive amount of time, and when the bus is held high during a transaction.

If the bus level remains low for at least 3.0 ms, all devices interpret the low bus level as a Global Reset signal, release control of the bus, and reset themselves.

If a command or data transaction is incomplete, the ADB bus stays high beyond the maximum bit-cell time. In this case, all devices ignore the command and wait for an Attention signal.

ADB peripheral devices

All peripheral devices on the ADB are slaves; only the computer can send commands. ADB devices transmit data on the bus only after receiving a Talk command from the computer.

Each ADB device is required to have

- memory in which to store data

- a default ADB address
- a default Device Handler ID
- the ability to detect and respond to bus collisions
- the ability to assert a Service Request signal

In addition, some ADB devices have more than one functional mode; the device driver can select the mode by assigning a new Device Handler ID to the device.

The following sections describe each of these features of an ADB device.

ADB device registers

An ADB device may have up to four locations in which to store data, referred to as registers 0 through 3. Each Talk or Listen command is directed to a specific register. The following paragraphs describe the use of each of these registers.

Register 0

Register 0 is a data register. When the ADB Manager polls devices to determine which one sent a Service Request signal, it does so by sending a Talk Register 0 command to each device in turn. Therefore, it is essential that a device have data in register 0 when it requires service, even if the data of significance to the device handler is in another register.

The computer can send data to a device with a Listen Register 0 command if the device is designed to accept such data.

Table 8-4 shows the bits of register 0 as used in the Apple Standard Mouse. Tables 8-5 and 8-8 show the bits of register 0 as used in the Apple Standard Keyboard and Apple Extended Keyboard.

Register 1

Register 1 is a device-specific data register. This register can be used by the device for any data function.

Register 2

Register 2 is a device-specific data register. This register can be used by the device for any data function. For example, Tables 8-8 and 8-11 show the bits of register 2 as used by the Apple Standard Keyboard and the Apple Extended Keyboard.

Register 3

This register contains status and device identification information. A description of each bit in register 3 is shown in Table 8-13. For some devices, the computer can change the function of the device by addressing this register with a Listen Register 3 command. The functions of the bits in this register are described in more detail in the sections that follow.

■ **Table 8-13** Bits in device register 3

Bit	Description
15	Reserved; must be 0
14	Exceptional event, device specific; always 1 if not used
13	Service Request enable; 1 = enabled
12	Reserved; must be 0
11–8	Device address
7–0	Device Handler ID

ADB device addresses

Each ADB device has a default 4-bit bus address. Certain device types have specified default addresses, as shown in Table 8-14. For example, all relative-position devices, such as a mouse, power up at address $03. Most devices have movable addresses; that is, the computer can assign a new address to the device. This is necessary because when two devices (such as a mouse device and a relative-position graphics tablet) have the same default address, one must be moved to a new address. Currently, eight addresses are predefined or reserved, leaving eight default addresses available for other types of devices.

■ **Table 8-14** Device addresses

Address	Device type	Example
$00–$01	Reserved	–
$02	Encoded devices	Keyboard
$03	Relative devices	Mouse
$04	Absolute devices	Graphics tablet
$05–$07	Reserved	–
$08–$0F	Any other	

ADB Device Handler ID

The ADB Manager uses the 8-bit Device Handler ID—together with the default ADB address—to determine which device driver to call for a particular device. An ADB device driveris a system software routine that communicates with a particular ADB device or class of devices; ADB device drivers are located in 'ADBS' resources in the System file. At system startup, the Start Manager searches the System file for 'ADBS' resources, loads them into memory, and executes them. The Start Manager also reads register 3 in each ADB device, and places the device's default address and Device Handler ID into the ADB device table.

Each 'ADBS' resource includes a device type (corresponding to the Device Handler ID) and an ADB address (corresponding to the device's default address). In the case of more than one device on the ADB with the same default address, the Start Manager reassigns devices to new addresses until there are no more conflicts. (See the next section, "ADB Collision Detection.") The ADB device table contains both the original (default) address and the current address for each device. The ADB Manager uses the default ADB address and Device Handler ID to associate each device on the bus with a particular device driver.

Two or more devices can use the same device driver. For example, a relative-position graphics tablet could emulate a mouse, using the same default ADB address and Device Handler ID as used by a mouse, and providing the same information in response to Talk commands. In this case, both the mouse and the graphics tablet could be connected to the bus at the same time, and the ADB Manager would call the Mouse Driver when either device required service.

A device may have more than one functional mode; if it does, the device driver can use a Listen Register 3 command to change the Device Handler ID. If the device recognizes the new Device Handler ID, it changes functional modes accordingly. An example of such a device is a graphics tablet that can operate as either a relative-position device or as an absolute-position device; another example is a keyboard that can generate more than one set of key-down transition codes. For more information, see the description of the Device Handler ID command in the chapter on the Apple Desktop Bus in *Inside Macintosh*.

Certain Device Handler IDs are reserved; each device must respond in a prescribed manner to a Listen Register 3 command specifying these IDs, as shown in Table 8-15. One of these Device Handler IDs ($00) can also be returned by a device to indicate that the device failed a self-test. Other Device Handler IDs may be used for special functions, but any new Device Handler ID must be assigned by Apple Computer, Inc.

■ **Table 8-15** Device Handler IDs reserved for special functions

ID value	Function definition
$FF	As Listen Register 3 data, initiates a self-test in the device.
$FE	As Listen Register 3 data, instructs the device to change the address field to the new address sent by the computer if no collision has been detected.
$FD	As Listen Register 3 data, instructs the device to change the address field to the new address sent by the computer if the activator is pressed. (The activator is explained in the next section, "ADB Collision Detection.")
$00	As Listen Register 3 data, instructs the device to change the address and enable fields to the new values sent by the computer.
$00	As data sent in response to a Talk Register 3 command, indicates that the device failed a self-test.

Upon receiving a reserved Device Handler ID, the device immediately performs the specified function. The device does not store the reserved Device Handler ID in register 3; only device-defined handler codes are stored. All unrecognized Device Handler IDs are ignored.

ADB collision detection

Each device waits to transmit data until it detects a free bus; that is, until the bus remains high for the stop-bit-to-start-bit time shown in Table 8-12. If a device is attempting to bring the bus high (for example; to complete a bit cell) and another device forces the line low, or if another device starts to send data before the device is able to assert its start bit, then the device is said to have lost a collision. All ADB devices must be able to detect collisions. The losing device immediatedly stops transmitting and preserves the data that was being sent. A device sets an internal flag if it loses a collision. This flag is cleared the next time the device successfully transmits data without detecting a collision.

◆ *Note:* Because a device could fail to detect a collision with another device that operates on a nearly identical internal clock, each device should attempt to assert its start bit at a random time within the stop-bit-to-start-bit time shown in Table 8-12.

Because all ADB devices are assigned unique addresses during startup, and because a device sends data only in response to a Talk command that is addressed specifically to that device, collisions do not ordinarily occur. When the bus is first started up or reset, however, there might be more than one device on the bus with the same default address.

At startup, the Start Manager sends a Talk Register 3 command to each ADB address. If there is more than one device at an address, each device that loses the collision sets its internal collision flag and disables its movable address function. The Start Manager then sends a Listen Register 3 command to that address with a Device Handler ID of $FE and a new device address. The device that did not detect the collision is moved to the new address. This process is repeated until the response to a Talk Register 3 command at that address is a timeout; that is, until there are no more devices at that address. Then one of the devices is moved back to the default address, and the Start Manager goes on to the next address.

An ADB device may have a key or button—called an *activator*—that can be used by multi-user applications to identify and locate individual devices. The activator might be a special key on a keyboard, for example, or a button on a mouse. An application can display a message requesting a user to press the activator. The device driver can then relocate the device to a new address by issuing a Listen Register 3 command with a Device Handler ID of $FD.

ADB polling protocol

After all address conflicts have been resolved, the Start Manager turns control of the ADB over to the ADB Manager and the ADB transceiver. If there are no service requests pending, the ADB Manager sends a Talk Register 0 command to address $03 (the default active device, usually the mouse) and the ADB transceiver repeats this command every 11 ms.

To send a Service Request signal, a device waits until the end of a command and then holds the bus low for 140 to 260 µs. Because the next command is normally a Talk Register 0 command addressed to the active device, the active device does not have to assert a Service Request signal to send data to the computer.

If another device has data to send, it sends a Service Request signal. When the ADB transceiver receives the Service Request signal, it sends an interrupt request to the VIA, which sets to 0 bit 3 in VIA Data register B. When the operating system finds a 0 in this bit, it passes control to the ADB Manager, which sends a Talk Register 0 command to each device on the bus until it finds the one requiring service. When a device responds to the Talk Register 0 command, the ADB Manager looks in the ADB device table to determine which device driver to call, and then passes control to the appropriate device driver.

The last device to send data to the computer becomes the active device, and the ADB transceiver addresses Talk Register 0 commands to that device every 11 ms until some other device asserts a Service Request signal.

Controlling the ADB Service Request

It is possible to control the ability of a device to transmit a Service Request signal. To disable a device's ability to send a Service Request signal, set bit 13 in register 3 to 0 by using a Listen Register 3 command with a Device Handler ID of $00, as shown in Table 8-15. To enable the Service Request ability, set this bit to 1. You can use this feature to improve service request response time when there are several devices on the bus and not all of them are required for a particular application.